Engagement Stage: Recommendation Systems

After identifying and targeting potential new customers through look-alike or uplift modeling, and enhancing engagement through cross-selling and upselling, the next step is to implement recommendation systems. These systems aim to suggest relevant products and services to customers based on their past behaviors and preferences, further enhancing customer engagement and satisfaction.

Recommendation Systems

Objective:

- Personalize customer experience by recommending products and services that align with their preferences and needs.
- Increase product adoption and customer satisfaction through targeted recommendations.

Methodology:

1. **Data Preparation:**
    - Collect and preprocess customer data, including transaction history, product holdings, and interaction data.
    - Data sources include transactional logs, customer profiles, and engagement metrics.
2. **Attribute Selection:**
    - **Raw Attributes:**
        - Customer ID
        - Age
        - Income
        - Occupation
        - Credit score (CIBIL score)
        - Product holdings (e.g., savings account, loan, credit card)

    - Transaction history (frequency, recency, monetary value)
    - Customer interactions (e.g., website visits, customer service interactions)
- **Derived Attributes:**
    - Product affinity score: Likelihood of interest in specific products based on past behavior and similar customer profiles.
    - Engagement score: Measure of overall engagement with the bank's services.
    - Spending patterns: Trends in spending and saving behavior.
    - Customer lifetime value (CLTV): Projected long-term value of the customer to the bank.
- **Target Variables:**
    - Recommended product uptake (binary: yes/no)
    - Customer satisfaction score after recommendation (optional)

3. **Model Selection:**
    - **Recommendation System Types:**
        - **Collaborative Filtering:** Based on similarities between users or items.
        - **Content-Based Filtering:** Based on similarities in product attributes.
        - **Hybrid Models:** Combines collaborative and content-based approaches.
    - **Algorithms:** Matrix Factorization, k-Nearest Neighbors (k-NN), Singular Value Decomposition (SVD), Neural Collaborative Filtering.

Implementation Steps

1. **Data Collection and Preparation:**

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, Standar
dScaler

# Sample customer data
```

```
data = {
    'customer_id': [1, 2, 3, 4, 5],
    'age': [25, 45, 35, 50, 23],
    'income': [50000, 120000, 75000, 100000, 45000],
    'cibil_score': [700, 800, 750, 780, 690],
    'product_holdings': ['savings,loan', 'savings,credit
card', 'savings,loan', 'savings', 'loan'],
    'transaction_frequency': [15, 50, 25, 30, 10],
    'average_transaction_value': [2000, 5000, 3000, 400
0, 1500],
    'engagement_score': [3, 5, 4, 4, 2]
}
df = pd.DataFrame(data)

# Encode categorical variables
encoder = OneHotEncoder()
encoded_products = encoder.fit_transform(df[['product_ho
ldings']]).toarray()
df = df.join(pd.DataFrame(encoded_products, columns=enco
der.get_feature_names_out(['product_holdings']))).drop
('product_holdings', axis=1)

# Standardize numerical features
scaler = StandardScaler()
numerical_features = ['age', 'income', 'cibil_score', 't
ransaction_frequency', 'average_transaction_value', 'eng
agement_score']
df[numerical_features] = scaler.fit_transform(df[numeric
al_features])
```

2. **Collaborative Filtering Example:**

   Matrix Factorization (using Singular Value Decomposition - SVD):

```
from surprise import Dataset, Reader, SVD
from surprise.model_selection import train_test_split, c
ross_validate

# Sample interaction data (user_id, item_id, rating)
```

```
interaction_data = {
    'user_id': [1, 1, 1, 2, 2, 3, 3, 4, 4, 5],
    'item_id': [101, 102, 103, 101, 103, 102, 104, 101,
104, 103],
    'rating': [5, 4, 3, 4, 5, 3, 4, 2, 5, 3]
}
interaction_df = pd.DataFrame(interaction_data)

# Convert data into Surprise format
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(interaction_df[['user_id',
'item_id', 'rating']], reader)

# Train-test split
trainset, testset = train_test_split(data, test_size=0.
2)

# Train SVD model
svd = SVD()
svd.fit(trainset)

# Evaluate model
predictions = svd.test(testset)
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=
5, verbose=True)
```

3. **Content-Based Filtering Example:**

   Using TF-IDF for Product Descriptions:

```
from sklearn.feature_extraction.text import TfidfVectori
zer
from sklearn.metrics.pairwise import linear_kernel

# Sample product descriptions
products = {
    'product_id': [101, 102, 103, 104],
    'description': [
        "Savings account with high interest rates",
```

```
        "Personal loan with flexible repayment options",
        "Credit card with cashback offers",
        "Home loan with low interest rates"
    ]
}
product_df = pd.DataFrame(products)

# Compute TF-IDF matrix
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(product_df['descripti
on'])

# Compute similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

# Function to get recommendations based on product descr
iption
def get_recommendations(product_id, cosine_sim=cosine_si
m):
    idx = product_df[product_df['product_id'] == product
_id].index[0]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1],
reverse=True)
    sim_scores = sim_scores[1:4]  # Get top 3 similar pr
oducts
    product_indices = [i[0] for i in sim_scores]
    return product_df['product_id'].iloc[product_indice
s]

# Example usage
print(get_recommendations(101))
```

4. **Hybrid Recommendation System Example:**

   Combining Collaborative and Content-Based Filtering:

```
# Assuming collaborative_filtering_predictions and conte
nt_based_predictions
```

```
# are dataframes with customer_id and recommended produc
t_id columns

# Merge predictions
hybrid_recommendations = pd.concat([collaborative_filter
ing_predictions, content_based_predictions]).drop_duplic
ates()

# Example to show recommendations for a specific custome
r
customer_id = 1
customer_recommendations = hybrid_recommendations[hybrid
_recommendations['customer_id'] == customer_id]
print(customer_recommendations)
```

Variables for Recommendation Systems

1. **Customer Attributes:**
   - **Customer ID:** Unique identifier for each customer.
   - **Age:** Influences financial needs and product preferences.
   - **Income:** Indicates financial capacity and potential product interest.
   - **Occupation:** Provides insights into lifestyle and financial requirements.
   - **Credit Score:** Reflects creditworthiness and financial behavior.
   - **Product Holdings:** Current products owned by the customer.
2. **Transactional and Behavioral Data:**
   - **Transaction Frequency:** Indicates engagement level with banking services.
   - **Average Transaction Value:** Suggests spending capacity and behavior.
   - **Engagement Score:** Measures overall interaction and activity with the bank.
   - **Customer Interactions:** Data from touchpoints like website visits and customer service.
3. **Derived Attributes:**

- **Product Affinity Score:** Calculated based on similarity to other customers who have purchased the product.
- **Spending Patterns:** Trends in spending and saving behavior.
- **Customer Lifetime Value (CLTV):** Estimated total value the customer will bring to the bank over time.

4. **Target Variables:**
    - **Recommended Product Uptake:** Binary indicator of whether the customer accepted the recommendation.
    - **Customer Satisfaction Score:** Optional measure of customer satisfaction post-recommendation.

Conclusion

By implementing a sophisticated recommendation system, the bank can personalize customer interactions, increase product adoption, and improve customer satisfaction. These systems leverage both collaborative and content-based approaches to provide tailored recommendations, thereby enhancing engagement and fostering long-term customer relationships.